Project *network*

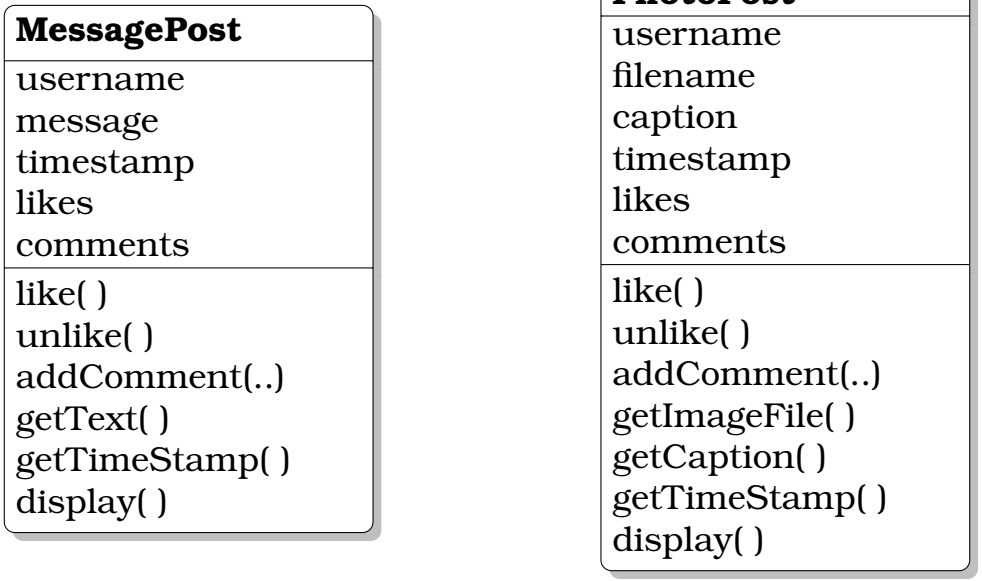
Gebaseerd op paragrafen **10.1-10.7**, **11.1-11.6** uit het boek.

We simuleren een sociaal netwerk

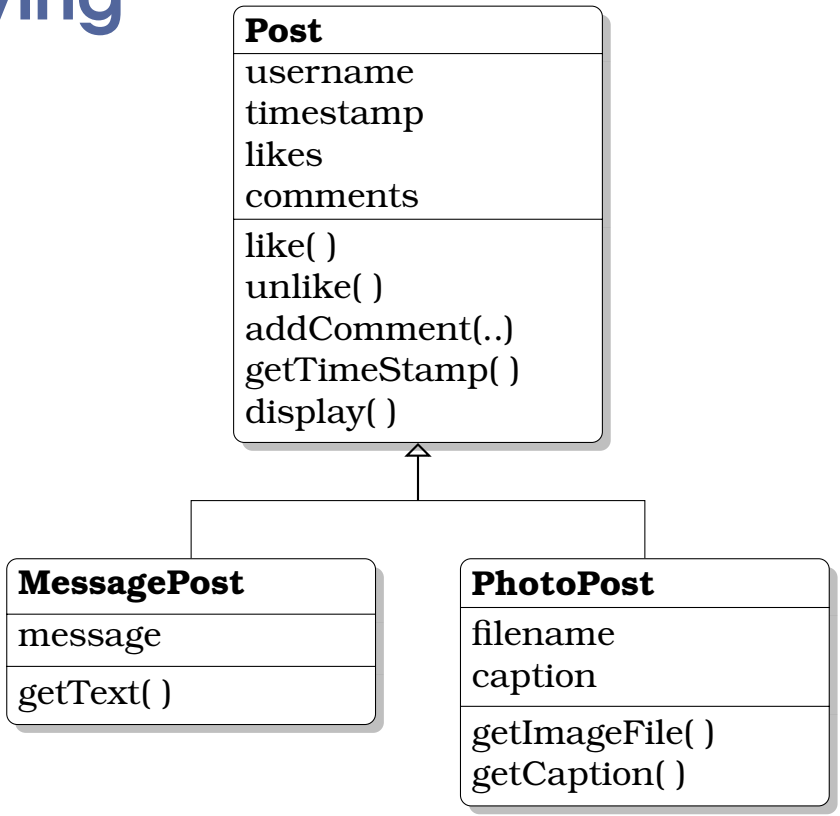
- Er zijn twee soorten berichten: *tekstberichten* en *fotoberichten*, ...
- voorgesteld door de klassen *MessagePost* en *PhotoPost*
- We houden deze berichten bij in een object van de klasse *NewsFeed*, ...
- in twee afzonderlijke lijsten:

```
private List<MessagePost> messages = ...;
private List<PhotoPost> photos = ...;
```

MessagePost en PhotoPost



Overerving


Post
username
timestamp
likes
comments
like( )
unlike( )
addComment(..)
getTimeStamp( )
display( )
MessagePost
message
getText( )
PhotoPost
filename
caption
getImageFile( )
getCaption( )

Overerving — terminologie

PhotoPost en *MessagePost* . . .

- *erven* van *Post*
- zijn *deelklassen* van *Post*
- zijn *subklassen* van *Post*
- *breiden Post uit*

Post

- is een *bovenklasse* van *PhotoPost* en *MessagePost*
- is een *superklasse* van *PhotoPost* en *MessagePost*

In Java:

```
public class PhotoPost extends Post {
    ...
}
```

Overerving — constructor

- De initializatie van een klasse begint *altijd* met de initializatie van haar bovenklasse.
- M.a.w., de constructor van een klasse moet *eerst* de constructor oproepen van de bovenklasse, ...
- met een speciale notatie: `super (...)`
- Als deze oproep ontbreekt, veronderstelt Java een impliciete `super()`

```
public MessagePost (String author, String message) {
  super (author);
  this.message = message;
}
```

Post heeft volgende constructor:

```
public Post (String author) { ... }
```

Overerving — protected

Velden van bovenklasse zijn niet toegankelijk voor klasse — want ze zijn *private*. Oplossing:

- Voorzie *getters* en *setters* voor deze velden, . . .
- of maak ze `protected`

```
protected int aantal;
```

Ook methoden kunnen protected zijn.

Ook een constructor kan protected zijn:

```
protected Post (String author) { ... }
```

Op die manier kunnen geen objecten van de klasse zelf worden aangemaakt — alleen van deelklassen.

Polymorfisme

Overal waar een object van een klasse is toegelaten, mag ook een object van een *deelklasse* gebruikt worden.

- Als inhoud van een variabele

```
Post post = new MessagePost ("KC", "Hallo!");
List<String> lijst = new ArrayList<>( );
```

- Als element van een array of lijst

```
List<Post> posts = ...
MessagePost messagePost = ...
...
posts.add (messagePost);
```

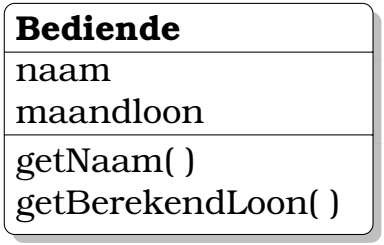
- Als parameter van een methode

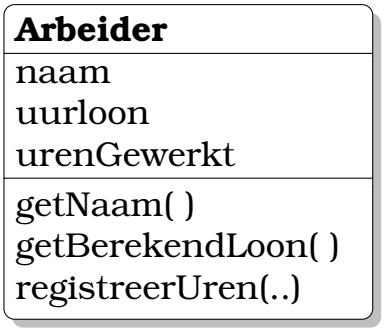
```
posts.add (messagePost);
```

- Enz...

Arbeiders en bedienden

Opgave Pas het project *personeel-v1* aan zodat het gebruik maakt van overerving.

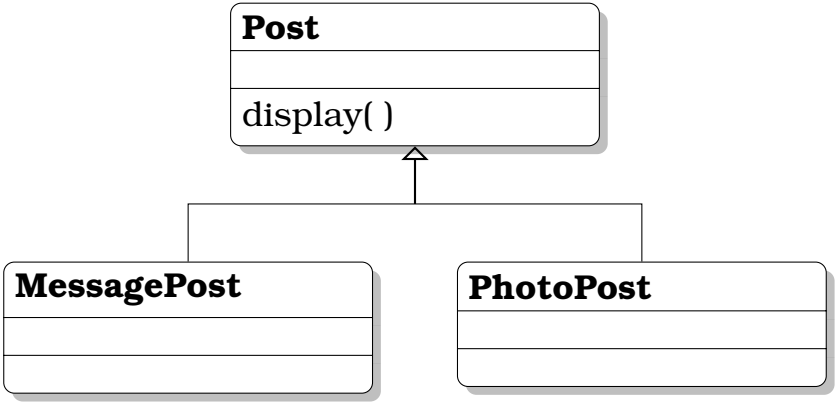




Opmerking. Er treedt een probleem op wanneer je *getBerekendLoon( )* naar een bovenklasse verplaatst. Wat is dit probleem? Los dit op door voorlopig voor iedereen een loon van 1000 EURO te berekenen.

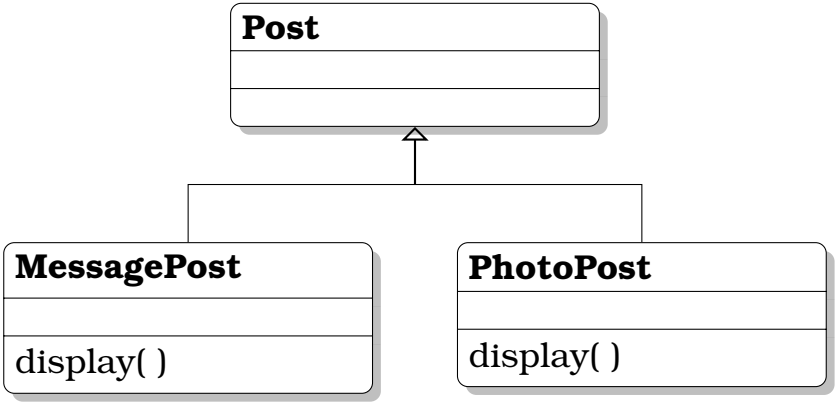
Het `display`-probleem

We willen dat tekstberichten en fotoberichten op een (lichtjes) andere manier worden afgebeeld.



Post heeft echter geen toegang tot velden van *MessagePost* of *PhotoPost*.

Het `display`-probleem — eerste poging

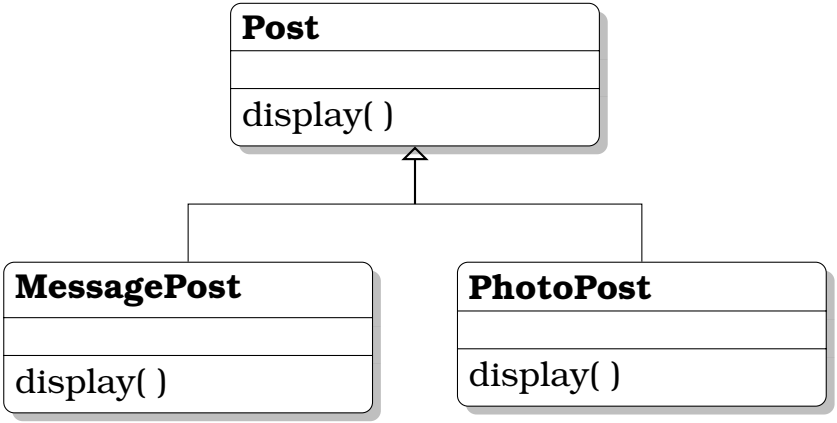


Compileert niet

```
for (Post post : posts) {
   post.display ();
   System.out.println ();
}
```

“cannot find symbol — method display()”

Het `display`-probleem — tweede poging



Welke van de drie methoden *display* worden opgeroepen?

- Dit wordt bepaald door de *klasse* waartoe het object behoort,
- Niet door het *type* van de variabele waarin het object zich bevindt.

Type vs. klasse

Type

- Van een *variabele*, *parameter* of *functie*
- Alleen belangrijk *at compile time*
- Bepaalt of een methode mag opgeroepen worden op een bepaalde variabele

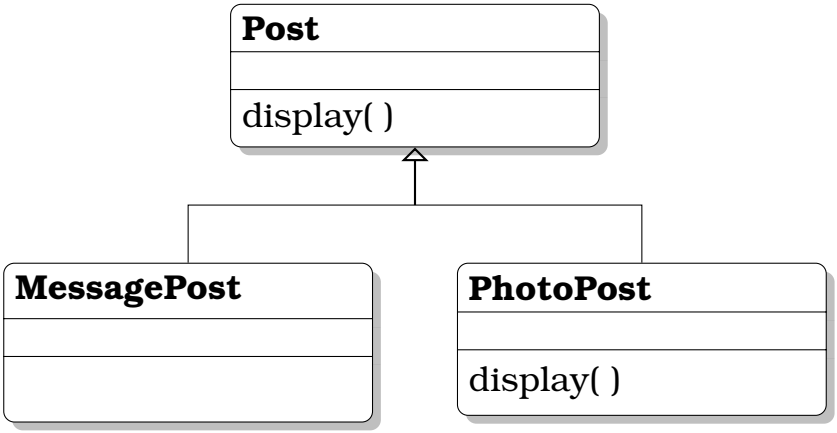
Klasse

- Van een *object*
- Wordt bepaald op het moment dat het object wordt aangemaakt met *new*
- Vooral belangrijk tijdens *run time*

Opmerking: Alternatieve terminologie:

- type = *statisch* type,
- klasse = *dynamisch* type

Het `display`-probleem — nog een versie



Wat gebeurt er als we onderstaande code proberen uitvoeren?

```
for (Post post : posts) {
   post.display ();
   System.out.println ();
}
```

De `super`-aanroep in methodes

Je kan ook de *superklasseversie* van een methode oproepen in de *subklasse*.

```
public class MessagePost {

    public void display() {
        super.display ();
        System.out.println(message);
    }
    ...
}
```

Arbeiders en bedienden — vervolg

- Hou één lijst bij van personeelsleden in plaats van afzonderlijke lijst voor bedienden en arbeiders
- Zorg dat *getBerekendLoon( )* doet wat het moet doen (m.a.w., niet hetzelfde voor arbeiders en bedienden).
- Voeg een methode *willekeurigePersoon* toe aan *Tester* en gebruik deze methode in *willekeurigeBerekening*.

Aangepast klassendiagram